DEVOPS SHACK

KUBERNETES PART-1

What is Kubernetes?

Kubernetes, often abbreviated as **K8s**, is an open-source container orchestration platform developed by Google and later donated to the **Cloud Native Computing Foundation (CNCF)**. It provides a framework for automating the deployment, scaling, and management of containerized applications.

At its core, Kubernetes solves the complexity of running applications at scale by abstracting the underlying infrastructure, making it easier to manage containers in a distributed environment. It handles a wide range of tasks like:

- **Automated Rollouts and Rollbacks**
- **Self-healing**
- **Service discovery**
- **Storage orchestration**
- **Load balancing**
- **Horizontal scaling**
- **Batch execution**

Key Components of Kubernetes:

1. **Cluster**:
    - A Kubernetes cluster consists of a group of **nodes** (physical or virtual machines) that run containerized applications. It has one or more master nodes and multiple worker nodes.
2. **Master Node**:
    - The control plane of Kubernetes responsible for managing the state of the cluster. It ensures the right number of containers are running, handles scheduling, and monitors the overall health of the system.

Key components:

   - **API Server**: The front end of the Kubernetes control plane, handling all API requests.
   - **Scheduler**: Decides where to run newly created containers.
   - **Controller Manager**: Ensures the desired state of the cluster is maintained.
   - **etcd**: Distributed key-value store that holds the state of the cluster.
3. **Worker Nodes**:
   - Each worker node is responsible for running the actual application containers. The nodes are managed by the master, which handles scheduling containers (known as **Pods**) onto the nodes.
4. **Pods**:
   - A **Pod** is the smallest deployable unit in Kubernetes. A pod can contain one or more containers that share storage, networking, and a lifecycle.
5. **Services**:
   - A **Service** in Kubernetes is an abstraction that defines a logical set of Pods and a policy by which to access them. It provides a stable IP address and DNS name to connect to applications running in different Pods.
6. **Namespaces**:
   - **Namespaces** are a way to divide cluster resources between multiple users or groups. Namespaces help in organizing objects and avoid name collisions in large Kubernetes clusters.
7. **Controllers**:
   - Controllers in Kubernetes are control loops that watch the state of your cluster, then make or request changes when the actual state doesn't match the desired state.

Why Do We Need Kubernetes?

Before understanding why Kubernetes is needed, it’s important to grasp the challenges of running applications in a **microservices** architecture.

Traditional Deployment Challenges:

- **Server Management**: Without Kubernetes, every application needs to be manually deployed on servers, which requires manual management of server health, load balancing, and scaling.
- **Resource Utilization**: Traditional deployment methods often underutilize hardware resources, as servers would either be over-provisioned or under-provisioned.
- **Scaling**: Manually scaling applications up or down based on user load is inefficient and error-prone.
- **Fault Tolerance**: Managing failures in applications, like restarting services or migrating them to healthy nodes, is complex without automation.
- **Configuration Management**: Tracking application versions and configurations across environments is cumbersome.

Benefits of Kubernetes:

1. **Container Orchestration**:
   - Kubernetes automates the deployment, management, and scaling of containerized applications. Containers, which are lightweight and portable, allow developers to package applications with all necessary dependencies. However, managing containers manually is challenging. Kubernetes simplifies this process by providing a unified platform to orchestrate and manage containers.
2. **Automated Scaling**:
   - Kubernetes can automatically scale applications up or down based on traffic or resource usage. It monitors the CPU and memory usage of containers and scales them dynamically to ensure optimal performance. This is achieved using Horizontal Pod Autoscalers (HPAs).
3. **Self-healing**:
   - Kubernetes automatically replaces and reschedules containers that fail, crash, or are unresponsive. If a node goes down, Kubernetes will restart the containers on a healthy node. It ensures high availability of applications by maintaining the desired number of healthy Pods.
4. **Load Balancing**:
   - Kubernetes provides built-in load balancing for services. When traffic to a service increases, Kubernetes spreads the load evenly across all Pods that provide the service, ensuring high availability and efficient resource utilization.
5. **Service Discovery**:
   - Applications running inside Kubernetes can automatically discover each other without needing complex networking configurations. Kubernetes provides DNS resolution for services, allowing easy communication between services.
6. **Declarative Configuration and Desired State**:
   - Kubernetes operates on a **desired state** model. Users declare the desired state of their applications, such as how many Pods should be running, what image to use, etc. Kubernetes continuously monitors the cluster to ensure the actual state matches the desired state, and takes actions to correct any deviations.
7. **Rolling Updates and Rollbacks**:
   - Kubernetes supports seamless updates to applications with zero downtime. With **rolling updates**, old versions of containers are gradually replaced with new ones, ensuring that the application remains available throughout the update process. In case of failures, Kubernetes can automatically rollback to the previous version.
8. **Infrastructure Abstraction**:
   - Kubernetes abstracts away the underlying infrastructure, making it possible to run applications on any cloud platform (AWS, Azure, GCP) or even on-premises. This helps avoid vendor lock-in and provides flexibility in terms of deployment strategies.

9. **Efficient Resource Utilization**:
    - Kubernetes intelligently schedules containers to ensure that the underlying hardware is used efficiently. It prevents resource wastage by packing containers onto nodes and redistributing containers when needed.
10. **Multi-cloud and Hybrid Cloud**:

- Kubernetes supports hybrid cloud and multi-cloud environments, allowing organizations to run their applications across multiple cloud providers or on both on-premises and cloud infrastructure. This increases flexibility and reduces dependency on a single cloud provider.

11. **CI/CD Integration**:

- Kubernetes works well with continuous integration/continuous deployment (CI/CD) pipelines, automating the deployment of applications after each code change. Integration with tools like Jenkins, GitLab CI, and others helps in smooth automation and application delivery.

Kubernetes Use Cases:

1. **Microservices Architecture**:
    - Kubernetes is particularly useful for applications following a **microservices architecture**, where different components of an application are isolated in different containers. Kubernetes allows you to easily manage, scale, and deploy these containers.
2. **High Availability Applications**:
    - Applications requiring **high availability** benefit from Kubernetes' automated scaling, self-healing, and failover capabilities. Kubernetes ensures that applications remain available even during node failures or updates.
3. **DevOps Automation**:
    - Kubernetes integrates well with DevOps practices, facilitating infrastructure as code (IaC), continuous deployment, and automated testing.
4. **Multi-tenant Environments**:
    - Organizations can run multiple projects or environments in the same Kubernetes cluster by leveraging **Namespaces** and **Role-Based Access Control (RBAC)** to isolate and manage resources efficiently.

When Not to Use Kubernetes:

While Kubernetes is highly powerful, it may not always be the best fit for every situation:

- **Small Applications**: For simple, small-scale applications, Kubernetes can be overkill, as it adds complexity.
- **Lack of Expertise**: Kubernetes has a steep learning curve, and without proper knowledge, it may become challenging to maintain a Kubernetes cluster.

- **Monolithic Applications**: If your application is monolithic and not containerized, Kubernetes might not bring many benefits.